



Programación Python para BigData

Lección 8: Apache Spark con PySpark
[2/2] (con hadoop)

ÍNDICE

Lección 8: Apache Spark con PySpark [2/2] (con hadoop)	3
Presentación y objetivos	3
1. Spark MLib	4
2. Spark GraphX.....	13
3. Diferencia entre Hadoop y Spark	27
4. Puntos claves.....	32

Lección 8: Apache Spark con PySpark [2/2] (con hadoop)

PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos a trabajar con Spark usando una librería de Python como es PySpark, para ello usaremos una imagen de Docker usada en la lección anterior y veremos los distintos usos con los que podemos trabajar.



Objetivos

- | Conocer el uso de Spark MLib
- | Conocer el uso de Spark GraphX
- | Breve introducción a Hadoop

1. SPARK MLIB

Para realizar esta parte usaremos el train.csv del dataset Titanic usado en lecciones anteriores.

Lo primero que debemos hacer es crear la sesión para Spark con nombre de la aplicación titanic cargando el csv train.csv:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('titanic').getOrCreate()

df = spark.read.csv('train.csv', header = True).cache()
```

Para visualizarlo usamos:

```
df.show(10)
```

MLib usando Titanic dataset

Crear la sesión de spark con nombre de la aplicación titanic y cargamos el dataset:

```
In [6]: from pyspark.sql import SparkSession
from pyspark.sql.types import StructType

spark = SparkSession.builder.appName('titanic').getOrCreate()

df = spark.read.csv('train.csv', header = True).cache()
```

```
In [7]: df.show(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	0	3	Braund, Mr. Owen ...	male	22	1	0	A/5 21171	7.25	null	S
2	2	1	1	Cumings, Mrs. Joh...	female	38	1	0	PC 17599	71.2833	C85	C
3	3	1	3	Heikkinen, Miss. ...	female	26	0	0	STON/O2. 3101282	7.925	null	S
4	4	1	1	Futrelle, Mrs. Ja...	female	35	1	0	113803	53.1	C123	S
5	5	0	3	Allen, Mr. Willia...	male	35	0	0	373450	8.05	null	S
6	6	0	3	Moran, Mr. James	male	null	0	0	330877	8.4583	null	Q
7	7	0	1	McCarthy, Mr. Tim...	male	54	0	0	17463	51.8625	E46	S
8	8	0	3	Palsson, Master. ...	male	2	3	1	349909	21.075	null	S
9	9	1	3	Johnson, Mrs. Osc...	female	27	0	2	347742	11.1333	null	S
10	10	1	2	Nasser, Mrs. Nich...	female	14	1	0	237736	30.0708	null	C

only showing top 10 rows

Figura 1.1 Crear la sesión en Spark y cargar el csv

Convertimos a pandas si queremos ver el dataframe:

```
df.toPandas()
```

```
In [8]: df.toPandas()
```

```
Out[8]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	None	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	None	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	None	S
...
886	887	0	2	Montvila, Rev. Juozas	male	27	0	0	211536	13	None	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19	0	0	112053	30	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	None	1	2	W/C. 6607	23.45	None	S
889	890	1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32	0	0	370376	7.75	None	Q

891 rows x 12 columns

Figura 1.2 Crear en dataframe los datos

Ver el número de filas que tiene

```
df.count()
```

```
In [9]: # Ver el número de filas que tiene
df.count()
```

```
Out[9]: 891
```

Figura 1.3 Números de datos del dataframe

Mostrar el nombre de las columnas:

```
df.columns
```

```
In [10]: # Mostrar el nombre de las columnas:
df.columns
```

```
Out[10]: ['PassengerId',
'Survived',
'Pclass',
'Name',
'Sex',
'Age',
'SibSp',
'Parch',
'Ticket',
'Fare',
'Cabin',
'Embarked']
```

Figura 1.4 Mostrar columnas del dataframe

Ver de qué tipo son las columnas para transformarlas en cada caso

```
df.dtypes
```

```
In [11]: # Ver de que tipo son las columnas para transformarlas en cada caso
df.dtypes

Out[11]: [('PassengerId', 'string'),
          ('Survived', 'string'),
          ('Pclass', 'string'),
          ('Name', 'string'),
          ('Sex', 'string'),
          ('Age', 'string'),
          ('SibSp', 'string'),
          ('Parch', 'string'),
          ('Ticket', 'string'),
          ('Fare', 'string'),
          ('Cabin', 'string'),
          ('Embarked', 'string')]
```

Figura 1.5 Mostrar de que tipo son los datos

Observamos que: 'Survived', 'Pclass', 'Age' y 'Fare' deberían de ser números más adelante los modificaremos.

Ver la descripción de nuestro dataset

```
df.describe().toPandas()
```

```
In [12]: # ver la descripción de nuestro dataset
df.describe().toPandas()
```

	summary	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
0	count	891	891	891	891	891	714	891
1	mean	446.0	0.3838383838383838	2.308641975308642	None	None	29.69911764705882	0.5230078563411896
2	stddev	257.3538420152301	0.48659245426485753	0.8360712409770491	None	None	14.526497332334035	1.1027434322934315
3	min	1	0	1	"Andersson, Mr. August Edvard ("Wennerstrom")"	female	0.42	0
4	max	99	1	3	van Melkebeke, Mr. Philemon	male	9	8

Figura 1.6 Resumen de los datos

Mediante sql modificamos las columnas que son string en floats:

```
from pyspark.sql.functions import col

dataset = df.select(col('Survived').cast('float'),
                    col('Pclass').cast('float'),
                    col('Sex'),
                    col('Age').cast('float'),
                    col('Fare').cast('float'),
                    col('Embarked')
                    )

dataset.show()
```

```
In [13]: from pyspark.sql.functions import col

dataset = df.select(col('Survived').cast('float'),
                    col('Pclass').cast('float'),
                    col('Sex'),
                    col('Age').cast('float'),
                    col('Fare').cast('float'),
                    col('Embarked'))

dataset.show()
```

Survived	Pclass	Sex	Age	Fare	Embarked
0.0	3.0	male	22.0	7.25	S
1.0	1.0	female	38.0	71.2833	C
1.0	3.0	female	26.0	7.925	S
1.0	1.0	female	35.0	53.1	S
0.0	3.0	male	35.0	8.05	S
0.0	3.0	male	null	8.4583	Q
0.0	1.0	male	54.0	51.8625	S
0.0	3.0	male	2.0	21.075	S
1.0	3.0	female	27.0	11.1333	S
1.0	2.0	female	14.0	30.0708	C
1.0	3.0	female	4.0	16.7	S
1.0	1.0	female	58.0	26.55	S
0.0	3.0	male	20.0	8.05	S
0.0	3.0	male	39.0	31.275	S
0.0	3.0	female	14.0	7.8542	S
1.0	2.0	female	55.0	16.0	S

Figura 1.7 Seleccionar columnas

Buscamos los datos nulos:

```
from pyspark.sql.functions import isnull, when, count, col

dataset.select([count(when(isnull(c), c)).alias(c) for c in
dataset.columns]).show()
```

```
In [14]: from pyspark.sql.functions import isnull, when, count, col

dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

Survived	Pclass	Sex	Age	Fare	Embarked
0	0	0	177	0	2

Figura 1.8 Eliminación de datos nulos

Reemplazar por donde venga ? por None y eliminarlos:

```
dataset = dataset.replace('?', None).dropna(how='any')
```

```
In [15]: dataset = dataset.replace('?', None).dropna(how='any')
```

Figura 1.9 Reemplazar valores y eliminarlos a posteriores

Crear columnas nuevas: Gender por 0 y 1 en el caso de mujer y hombre y Boarded donde embarcaron 0, 1 y 2, mediante el uso de StringIndexer que nos transforma en categóricas:

```
from pyspark.ml.feature import StringIndexer

dataset = StringIndexer(inputCol='Sex', outputCol='Gender',
                        handleInvalid='keep').fit(dataset).transform(dataset)
dataset = StringIndexer(inputCol='Embarked', outputCol='Boarded',
                        handleInvalid='keep').fit(dataset).transform(dataset)
dataset.show()
```

Crear columnas nuevas: Gender por 0 y 1 en el caso de mujer y hombre y Boarded donde embarcaron 0,1 y 2

```
In [16]: from pyspark.ml.feature import StringIndexer

dataset = StringIndexer(inputCol='Sex', outputCol='Gender', handleInvalid='keep').fit(dataset).transform(dataset)
dataset = StringIndexer(inputCol='Embarked', outputCol='Boarded', handleInvalid='keep').fit(dataset).transform(dataset)
dataset.show()
```

Survived	Pclass	Sex	Age	Fare	Embarked	Gender	Boarded
0.0	3.0	male	22.0	7.25	S	0.0	0.0
1.0	1.0	female	38.0	71.2833	C	1.0	1.0
1.0	3.0	female	26.0	7.925	S	1.0	0.0
1.0	1.0	female	35.0	53.1	S	1.0	0.0
0.0	3.0	male	35.0	8.05	S	0.0	0.0
0.0	1.0	male	54.0	51.8625	S	0.0	0.0
0.0	3.0	male	2.0	21.075	S	0.0	0.0
1.0	3.0	female	27.0	11.1333	S	1.0	0.0
1.0	2.0	female	14.0	30.0708	C	1.0	1.0
1.0	3.0	female	4.0	16.7	S	1.0	0.0
1.0	1.0	female	58.0	26.55	S	1.0	0.0
0.0	3.0	male	20.0	8.05	S	0.0	0.0
0.0	3.0	male	39.0	31.275	S	0.0	0.0
0.0	3.0	female	14.0	7.8542	S	1.0	0.0
1.0	2.0	female	55.0	16.0	S	1.0	0.0
0.0	3.0	male	2.0	29.125	Q	0.0	2.0
0.0	3.0	female	31.0	18.0	S	1.0	0.0

Figura 1.10 Transformación de variables categóricas

Comprobación de que es correcto el formato:

dataset.dtypes

```
In [17]: dataset.dtypes
Out[17]: [('Survived', 'float'),
          ('Pclass', 'float'),
          ('Sex', 'string'),
          ('Age', 'float'),
          ('Fare', 'float'),
          ('Embarked', 'string'),
          ('Gender', 'double'),
          ('Boarded', 'double')]
```

Figura 1.11 Mostrar de que tipo son los datos

Eliminar columnas innecesarias:

```
dataset = dataset.drop('Sex')
dataset = dataset.drop('Embarked')
dataset.show()
```



```
In [18]: # Eliminar columnas innecesarias
dataset = dataset.drop('Sex')
dataset = dataset.drop('Embarked')
dataset.show()
```

Survived	Pclass	Age	Fare	Gender	Boarded
0.0	3.0	22.0	7.25	0.0	0.0
1.0	1.0	38.0	71.2833	1.0	1.0
1.0	3.0	26.0	7.925	1.0	0.0
1.0	1.0	35.0	53.1	1.0	0.0
0.0	3.0	35.0	8.05	0.0	0.0
0.0	1.0	54.0	51.8625	0.0	0.0
0.0	3.0	2.0	21.075	0.0	0.0
1.0	3.0	27.0	11.1333	1.0	0.0
1.0	2.0	14.0	30.0708	1.0	1.0
1.0	3.0	4.0	16.7	1.0	0.0
1.0	1.0	58.0	26.55	1.0	0.0
0.0	3.0	20.0	8.05	0.0	0.0
0.0	3.0	39.0	31.275	0.0	0.0
0.0	3.0	14.0	7.8542	1.0	0.0
1.0	2.0	55.0	16.0	1.0	0.0
0.0	3.0	2.0	29.125	0.0	2.0
0.0	3.0	31.0	18.0	1.0	0.0
0.0	2.0	35.0	26.0	0.0	0.0
1.0	2.0	34.0	13.0	0.0	0.0
1.0	3.0	15.0	8.0292	1.0	2.0

only showing top 20 rows

Figura 1.12 Eliminar columnas innecesarias

Reunir todas las funciones con VectorAssembler

```
required_features = ['Pclass',
                    'Age',
                    'Fare',
                    'Gender',
                    'Boarded'
                    ]

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(dataset)

transformed_data.show(5)
```

Transformamos toda la información como objetos:

```
In [19]: # Reunir todas las funciones con VectorAssembler
required_features = ['Pclass',
                    'Age',
                    'Fare',
                    'Gender',
                    'Boarded'
                    ]

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(dataset)
```

```
In [20]: transformed_data.show(5)
```

Survived	Pclass	Age	Fare	Gender	Boarded	features
0.0	3.0	22.0	7.25	0.0	0.0	[3.0,22.0,7.25,0.0,0.0]
1.0	1.0	38.0	71.2833	1.0	1.0	[1.0,38.0,71.2833,1.0,1.0]
1.0	3.0	26.0	7.925	1.0	0.0	[3.0,26.0,7.925,1.0,0.0]
1.0	1.0	35.0	53.1	1.0	0.0	[1.0,35.0,53.0999,1.0,0.0]
0.0	3.0	35.0	8.05	0.0	0.0	[3.0,35.0,8.05,0.0,0.0]

only showing top 5 rows

Figura 1.13 Transformación en vector

División para entrenar el modelo:

```
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
```

División de entrenamiento:

```
In [22]: (training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
```

Figura 1.14 División para entrenar el modelo del csv train en dos train y test

Aplicar el modelo Decision Tree Classifier:

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(labelCol='Survived',
                             featuresCol='features',
                             maxDepth=5)
```

• Algoritmo DecisionTreeClassifier

```
In [29]: from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(labelCol='Survived',
                             featuresCol='features',
                             maxDepth=5)
```

Figura 1.15 Aplicar el algoritmo Decision Tree Classifier

Modelo:

```
model = dt.fit(training_data)
```

Predicción:

```
predictions = model.transform(test_data)

predictions.show(5)
```

```
In [30]: model = dt.fit(training_data)
```

```
In [31]: predictions = model.transform(test_data)
```

```
In [32]: predictions.show(5)
```

Survived	Pclass	Age	Fare	Gender	Boarded	features	rawPrediction	probability	prediction
0.0	1.0	29.0	66.6	0.0	0.0	[1.0, 29.0, 66.5999...]	[14.0, 19.0]	[0.42424242424242...]	1.0
0.0	1.0	36.0	40.125	0.0	1.0	[1.0, 36.0, 40.125, ...]	[14.0, 19.0]	[0.42424242424242...]	1.0
0.0	1.0	37.0	29.7	0.0	1.0	[1.0, 37.0, 29.7000...]	[14.0, 19.0]	[0.42424242424242...]	1.0
0.0	1.0	38.0	0.0	0.0	0.0	[5.0, 11.0, 1.0, 38.0]	[14.0, 19.0]	[0.42424242424242...]	1.0
0.0	1.0	42.0	52.0	0.0	0.0	[1.0, 42.0, 52.0, 0.0...]	[14.0, 19.0]	[0.42424242424242...]	1.0

only showing top 5 rows

Figura 1.16 Predicción del modelo

Evaluar el modelo obteniendo la precisión:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol='Survived',
predictionCol='prediction',
metricName='accuracy')

accuracy = evaluator.evaluate(predictions)
print('Test Accuracy = ', accuracy)
```

```
In [33]: # Evaluar el modelo
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol='Survived', predictionCol='prediction',
metricName='accuracy')

In [34]: accuracy = evaluator.evaluate(predictions)
print('Test Accuracy = ', accuracy)
Test Accuracy = 0.7956204379562044
```

Figura 1.17 Precisión del modelo

Se obtiene que nuestro modelo es preciso en un 0,79 de los casos.

Aplicar el modelo Gradient-boosted tree classifier:

```
from pyspark.ml.classification import GBTClassifier

# Entrenar el modelo GBT
gbt = GBTClassifier(labelCol="Survived", featuresCol="features", maxIter=10)
```

. Algoritmo Gradient-boosted tree classifier

```
In [35]: from pyspark.ml.classification import GBTClassifier

# Entrenar el modelo GBT
gbt = GBTClassifier(labelCol="Survived", featuresCol="features", maxIter=10)
```

Figura 1.18 Aplicar el algoritmo Gradient-Booted Tree Classifier

```
# Modelo:
modelGBT = gbt.fit(training_data)

# Hacer predicción.
predictionsGBT = modelGBT.transform(test_data)

# seleccionar filas de ejemplo para mostrar.
predictionsGBT.select("prediction", "probability", "features").show(5)
```

```
In [37]: # Modelo:
modelGBT = gbt.fit(training_data)

In [38]: # Hacer predicción.
predictionsGBT = modelGBT.transform(test_data)

In [39]: # seleccionar filas de ejemplo para mostrar.
predictionsGBT.select("prediction", "probability", "features").show(5)
```

prediction	probability	features
1.0	[0.44476061946898...	[1.0,29.0,66.5999...
1.0	[0.35066971222968...	[1.0,36.0,40.125...
1.0	[0.44246439315406...	[1.0,37.0,29.7000...
0.0	[0.76021883560221...	[5,[0,1],[1.0,38.0]]
1.0	[0.44476061946898...	[1.0,42.0,52.0,0...

only showing top 5 rows

```
21/08/20 09:59:18 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
21/08/20 09:59:18 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
```

Figura 1.19 Predicción del algoritmo Gradient-Booted Tree Classifier

Evaluar el modelo obteniendo la precisión y el error:

```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="Survived",
predictionCol="prediction",
metricName="accuracy")

accuracy = evaluator.evaluate(predictionsGBT)
print('Test Accuracy = ', accuracy)
print("Test Error = %g" % (1.0 - accuracy))
```

```
In [42]: # Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction",
metricName="accuracy")

accuracy = evaluator.evaluate(predictionsGBT)
print('Test Accuracy = ', accuracy)
print("Test Error = %g" % (1.0 - accuracy))

Test Accuracy = 0.781021897810219
Test Error = 0.218978
```

Figura 1.20 Precisión del algoritmo Gradient-Booted Tree Classifier

Resultado es de precisión 0,78

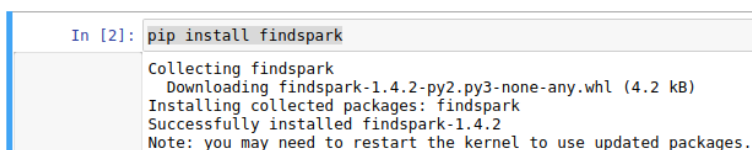
2. SPARK GRAPHX

GraphFrame admite el procesamiento general de gráficos, que es similar a la biblioteca GraphX de Apache Spark. Además, GraphFrames se basa en Spark DataFrames, que tiene las siguientes ventajas:

- | API de Python, Java y Scala: GraphFrames proporciona interfaces API comunes para los tres lenguajes. Es la primera vez que todos los algoritmos implementados en GraphX se pueden usar en Python y Java.
- | Consultas potentes: GraphFrames permite consultas breves, al igual que las consultas potentes en Spark SQL y DataFrame.
- | Guardar y cargar modelos de gráficos: GraphFrames es totalmente compatible con las fuentes de datos de estructura DataFrame, lo que permite el uso de Parquet, JSON y CSV familiares para leer y escribir gráficos.

Será necesario instalar:

```
pip install findspark
```



```
In [2]: pip install findspark
Collecting findspark
  Downloading findspark-1.4.2-py2.py3-none-any.whl (4.2 kB)
Installing collected packages: findspark
Successfully installed findspark-1.4.2
Note: you may need to restart the kernel to use updated packages.
```

Figura 2.1 Instalación de findspark

Para instalar graphframes:

```
import findspark
findspark.init()
import pyspark
import os

SUBMIT_ARGS = "--packages graphframes:graphframes:0.8.1-spark3.0-s_2.12 pyspark-shell"
os.environ["PYSPARK_SUBMIT_ARGS"] = SUBMIT_ARGS

conf = pyspark.SparkConf()
sc = pyspark.SparkContext(conf=conf)
print(sc._conf.getAll())
```

```
In [3]: import findspark
findspark.init()
import pyspark
import os

SUBMIT_ARGS = "--packages graphframes:graphframes:0.8.1-spark3.0-s_2.12 pyspark-shell"
os.environ["PYSPARK_SUBMIT_ARGS"] = SUBMIT_ARGS

conf = pyspark.SparkConf()
sc = pyspark.SparkContext(conf=conf)
print(sc._conf.getAll())

WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/spark-3.1.2-bin-hadoop3.2/jars/spark-unsafe_2.12-3.1.2.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release

:: loading settings :: url = jar:file:/usr/local/spark-3.1.2-bin-hadoop3.2/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml

Ivy Default Cache set to: /home/jovyan/.ivy2/cache
The jars for the packages stored in: /home/jovyan/.ivy2/jars
graphframes#graphframes added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-4758e8db-a4e9-4d28-afaa-2c5e5554cb61;1.0
  confs: [default]
    found graphframes#graphframes:0.8.1-spark3.0-s_2.12 in spark-packages
    found org.slf4j#slf4j-api:1.7.16 in central
downloading https://repos.spark-packages.org/graphframes/graphframes/0.8.1-spark3.0-s_2.12/graphframes-0.8.1-spark3.0-s_2.12.jar ...
[SUCCESSFUL] graphframes#graphframes:0.8.1-spark3.0-s_2.12!graphframes.jar (684ms)
:: resolution report :: resolve 5464ms :: artifacts dl 695ms
  :: modules in use:
    graphframes#graphframes:0.8.1-spark3.0-s_2.12 from spark-packages in [default]
    org.slf4j#slf4j-api:1.7.16 from central in [default]

  |-----|
  | conf | number | search | downloaded | evicted | | artifacts | |
|---|---|---|---|---|---|---|---|
  | default | 2 | 1 | 1 | 0 | | 2 | 1 |
  |-----|

:: retrieving :: org.apache.spark#spark-submit-parent-4758e8db-a4e9-4d28-afaa-2c5e5554cb61
  confs: [default]
  1 artifacts copied, 1 already retrieved (242kB/13ms)
21/08/20 13:48:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-j
ava classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

[('spark.executor.id', 'driver'), ('spark.app.initial.jar.urls', 'spark://bbcf1ddaafce:34557/jars/graphframes_g
raphframes-0.8.1-spark3.0-s_2.12.jar,spark://bbcf1ddaafce:34557/jars/org.slf4j_slf4j-api-1.7.16.jar'), ('spark.submi
t.pyFiles', '/home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.jar,/home/jovyan/.ivy2/jars/o
rg.slf4j_slf4j-api-1.7.16.jar'), ('spark.driver.host', 'bbcf1ddaafce'), ('spark.app.name', 'pyspark-shell'), ('spa
rk.driver.extraJavaOptions', '-Dio.netty.tryReflectionSetAccessible=true'), ('spark.jars', 'file:///home/jovyan/.i
vy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.jar,file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-1.7.1
6.jar'), ('spark.app.startTime', '1629467318211'), ('spark.driver.port', '34557'), ('spark.rdd.compress', 'True'),
('spark.app.initial.file.urls', 'file:///home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.jar,
file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-1.7.16.jar'), ('spark.serializer.objectStreamReset', '100'),
('spark.master', 'local[*]'), ('spark.submit.deployMode', 'client'), ('spark.executor.extraJavaOptions', '-Dio.net
ty.tryReflectionSetAccessible=true'), ('spark.files', 'file:///home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.
1-spark3.0-s_2.12.jar,file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-1.7.16.jar'), ('spark.repl.local.jars',
'file:///home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.jar,file:///home/jovyan/.ivy2/jars/o
rg.slf4j_slf4j-api-1.7.16.jar'), ('spark.ui.showConsoleProgress', 'true'), ('spark.app.id', 'local-162946731983
6')]
```

Figura 2.2 Instalación de GraphFrames

```
import sys
pyfiles = str(sc.getConf().get(u'spark.submit.pyFiles')).split(',')
sys.path.extend(pyfiles)
```

```
In [4]: import sys
        pyfiles = str(sc.getConf().get(u'spark.submit.pyFiles')).split(',')
        sys.path.extend(pyfiles)
```

Figura 2.3 Instalación de GraphFrames

Crear los dataframe con los cuales trabajamos:

```
from pyspark import *
from pyspark.sql import *
from graphframes import *

spark = SparkSession.builder.appName('graphFrames').getOrCreate()

# Crear los dataframes
v = spark.createDataFrame([("A", "ANA" , 350 ), ("B", "BERTO" , 360 ),
                           ("C", "CLARA" , 195 ), ("D", "DANIEL", 90),
                           ("E", "ERICA" , 90), ("F", "FRANCISCO" , 215 ),
                           ("G", "GERARDO", 30 ), ("H", "HERNANDO" , 25 ),
                           ("I", "INMA" , 25 ), ("J", "JUAN" , 20 )],
                           ["id", "name", "total_points"])

e=spark.createDataFrame([("A", "B", 60), ("B", "A", 60), ("A", "C", 50), ("D", "A", 100),
                           ("A", "D", 80), ("C", "I", 25), ("C", "J", 20), ("B", "F", 50),
                           ("F", "B", 60), ("F", "G", 110), ("F", "H", 25), ("B", "E", 90)
                           ], ["src", "dst", "relationship"])

# Construir el graph
g = GraphFrame(v,e)
```

```
In [5]: from pyspark import *
from pyspark.sql import *
from graphframes import *

spark = SparkSession.builder.appName('graphFrames').getOrCreate()

# Crear los dataframes
v = spark.createDataFrame([
    ("A", "ANA", 350),
    ("B", "BERTO", 360),
    ("C", "CLARA", 195),
    ("D", "DANIEL", 90),
    ("E", "ERICA", 90),
    ("F", "FRANCISCO", 215),
    ("G", "GERARDO", 30),
    ("H", "HERNANDO", 25),
    ("I", "INMA", 25),
    ("J", "JUAN", 20)
], ["id", "name", "total_points"])

e = spark.createDataFrame([
    ("A", "B", 60),
    ("B", "A", 60),
    ("A", "C", 50),
    ("D", "A", 100),
    ("A", "D", 80),
    ("C", "I", 25),
    ("C", "J", 20),
    ("B", "F", 50),
    ("F", "B", 60),
    ("F", "G", 110),
    ("F", "H", 25),
    ("B", "E", 90)
], ["src", "dst", "relationship"])

# Construir el graph
g = GraphFrame(v,e)
```

Figura 2.4 Crear sesión y cargar los dataframe vértice y borde

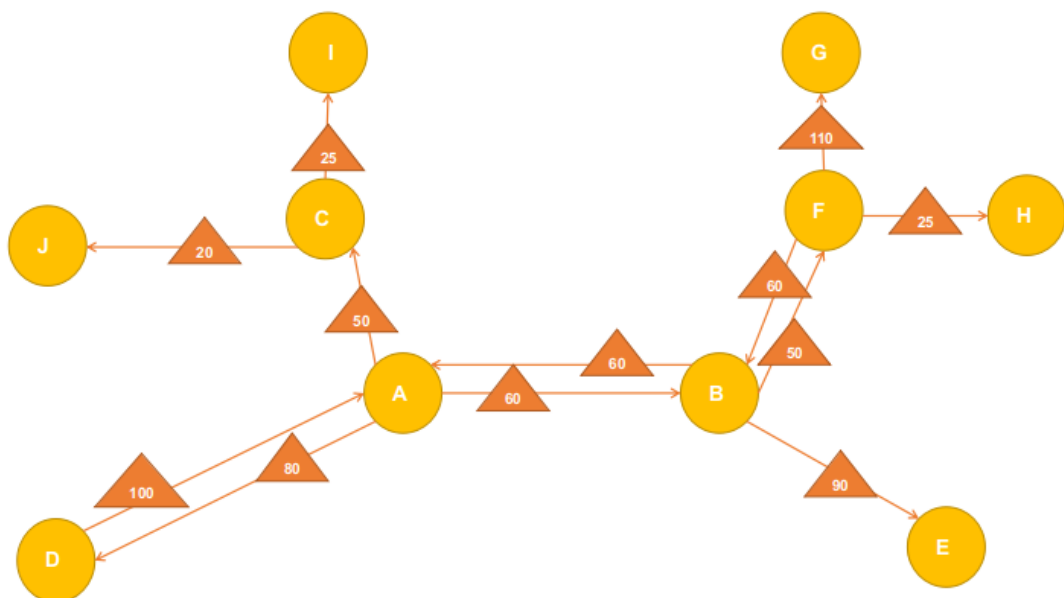


Figura 2.5 Mostrar el gráfico

Atributos básicos (vértices/edges):

Vértices:

```
verticesDF=g.vertices
```

```
edgesDF=g.edges
```

```
verticesDF.show()
```

Atributos básicos Graph:

- Definimos cual corresponde al vértice y cual al borde:

```
In [6]: verticesDF=g.vertices
edgesDF=g.edges
```

```
In [7]: verticesDF.show()
```

id	name	total_points
A	ANA	350
B	BERTO	360
C	CLARA	195
D	DANIEL	90
E	ERICA	90
F	FRANCISCO	215
G	GERARDO	30
H	HERNANDO	25
I	INMA	25
J	JUAN	20

Figura 2.6 Mostrar el vértice

```
edgesDF.show()
```

```
In [8]: edgesDF.show()
```

src	dst	relationship
A	B	60
B	A	60
A	C	50
D	A	100
A	D	80
C	I	25
C	J	20
B	F	50
F	B	60
F	G	110
F	H	25
B	E	90

Figura 2.7 Mostrar el borde

inDegrees, outDegrees y degrees:

- | InDegrees son las entradas al nodo.
- | outDegrees son las salidas del nodo
- | Degrees son todas las conexiones del nodo

```
inDegreeDF=g.inDegrees
```

```
outDegreeDF=g.outDegrees
```

```
degreeDF=g.degrees
```

```
inDegreeDF.sort(['inDegree'],ascending=[0]).show() # Sort and show
```

- inDegrees, outDegrees y degrees: inDegrees son las entradas del nodo, las outDegrees son las salidas del nodo y degrees son todas las conexiones por nodo.

```
In [9]: inDegreeDF=g.inDegrees
outDegreeDF=g.outDegrees
degreeDF=g.degrees

inDegreeDF.sort(['inDegree'],ascending=[0]).show() # Sort and show
```

[Stage 5:=====> (175 + 5) / 200]

id	inDegree
A	2
B	2
E	1
F	1
D	1
I	1
J	1
C	1
G	1
H	1

Figura 2.8 Mostrar los indegrees, outdegrees y degrees

```
outDegreeDF.sort(['outDegree'],ascending=[0]).show()
```

```
In [10]: outDegreeDF.sort(['outDegree'],ascending=[0]).show()
```

[Stage 7:=====> (173 + 5) / 200]

id	outDegree
B	3
A	3
F	3
C	2
D	1

Figura 2.9 Mostrar los indegrees, outdegrees y degrees

```
degreeDF.show()
```

```
In [11]: degreeDF.show() |
```

id	degree
F	4
E	1
B	5
D	2
C	3
J	1
A	5
G	1
I	1
H	1

Figura 2.10 Mostrar los indegrees, outdegrees y degrees

Estructura del graph

PageRank

Permite calcular los pesos para cada nodo o lo que es lo mismo asignar de forma numérica la relevancia de los nodos. Para ello ponemos:

```
PageRankResults = g.pageRank(resetProbability=0.15, tol=0.01)
```

```
PageRankResults.vertices.sort(['pagerank'],ascending=[0]).show()
```

PageRank

```
In [12]: PageRankResults = g.pageRank(resetProbability=0.15, tol=0.01)
# Lets sort it to see who are the most influential users.
PageRankResults.vertices.sort(['pagerank'],ascending=[0]).show()
```

[Stage 273:=====> (158 + 6) / 200]

id	name	total_points	pagerank
A	ANA	350	1.6755780131663474
B	BERTO	360	1.2302848386832057
D	DANIEL	90	0.99894394905248
C	CLARA	195	0.99894394905248
J	JUAN	20	0.9313848137403751
I	INMA	25	0.9313848137403751
F	FRANCISCO	215	0.8612020763296273
E	ERICA	90	0.8612020763296273
H	HERNANDO	25	0.7555377349527411
G	GERARDO	30	0.7555377349527411

Figura 2.11 Mostrar PageRank

PageRankResults.edges.show()

```
In [13]: # Lets see how PageRank has normalized the weights based on number of edges.
PageRankResults.edges.show()
```

src	dst	relationship	weight
C	J	20	0.5
A	B	60	0.3333333333333333
B	F	50	0.3333333333333333
C	I	25	0.5
F	H	25	0.3333333333333333
B	A	60	0.3333333333333333
F	G	110	0.3333333333333333
A	D	80	0.3333333333333333
D	A	100	1.0
F	B	60	0.3333333333333333
B	E	90	0.3333333333333333
A	C	50	0.3333333333333333

Figura 2.12 Mostrar PageRank

Label Propagation Algorithm (LPA)

Poder obtener la relación entre nodos obteniendo los grupos que lo forman, asignando una etiqueta a cada grupo.

```
result = g.labelPropagation(maxIter=5)
result.sort(['label'],ascending=[0]).show()
```

Label Propagation Algorithm (LPA)

Se observan las relaciones que se dan entre nodos clasificándolos en grupos:

```
In [14]: result = g.labelPropagation(maxIter=5)
result.sort(['label'],ascending=[0]).show()

[Stage 605:=====> (125 + 4) / 200]
```

id	name	total_points	label
C	CLARA	195	1391569403904
J	JUAN	20	764504178688
I	INMA	25	764504178688
F	FRANCISCO	215	420906795008
E	ERICA	90	420906795008
A	ANA	350	420906795008
D	DANIEL	90	171798691840
H	HERNANDO	25	171798691840
G	GERARDO	30	171798691840
B	BERTO	360	171798691840

Figura 2.13 Mostrar LPA

Conectados:

D,H,G, B | F, E, A | J, I | C

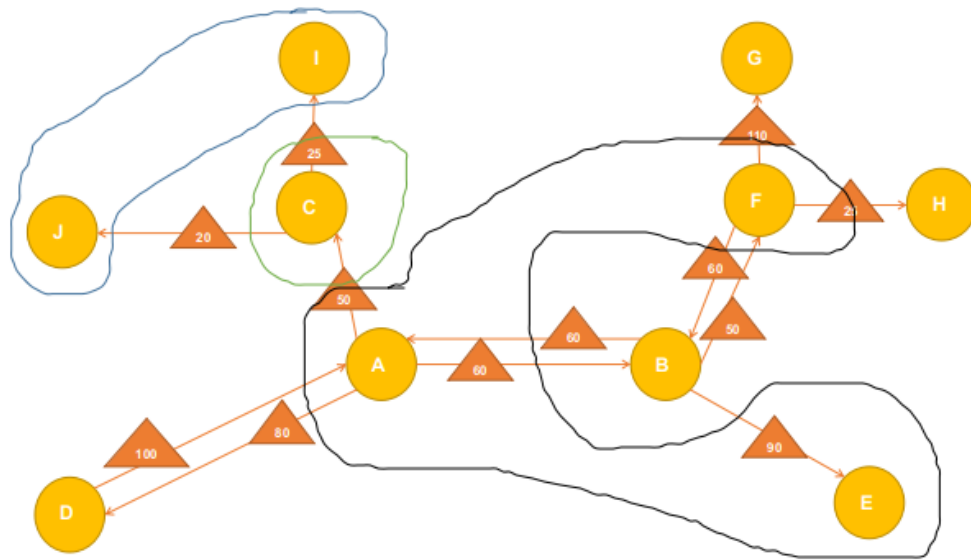


Figura 2.14 Mostrar LPA

Connected Components

Observar cuales la conexión entre los nodos, si hay nodos aislados o no.

```
# sc = sparkContext creado al inicio
sc.setCheckpointDir('graphframes_cps')
result = g.connectedComponents()
result.show()
```

```
Connected Components

In [15]: # sc = sparkContext creado al inicio
sc.setCheckpointDir('graphframes_cps')
result = g.connectedComponents()
result.show()

[Stage 742:=====] (178 + 4) / 200

+----+-----+-----+-----+
| id | name | total_points | component |
+----+-----+-----+-----+
| A | ANA | 350 | 171798691840 |
| B | BERTO | 360 | 171798691840 |
| C | CLARA | 195 | 171798691840 |
| D | DANIEL | 90 | 171798691840 |
| E | ERICA | 90 | 171798691840 |
| F | FRANCISCO | 215 | 171798691840 |
| G | GERARDO | 30 | 171798691840 |
| H | HERNANDO | 25 | 171798691840 |
| I | INMA | 25 | 171798691840 |
| J | JUAN | 20 | 171798691840 |
+----+-----+-----+-----+
```

Conclusión: Todos los datos estan conectados, por lo tanto forman un único clúster

Figura 2.15 Mostrar Componentes conectados

Todos están conectados.

Strongly Connected Components

Ver que nodos están fuertemente conectados.

```
result = g.stronglyConnectedComponents(maxIter=10)
result.show()
```

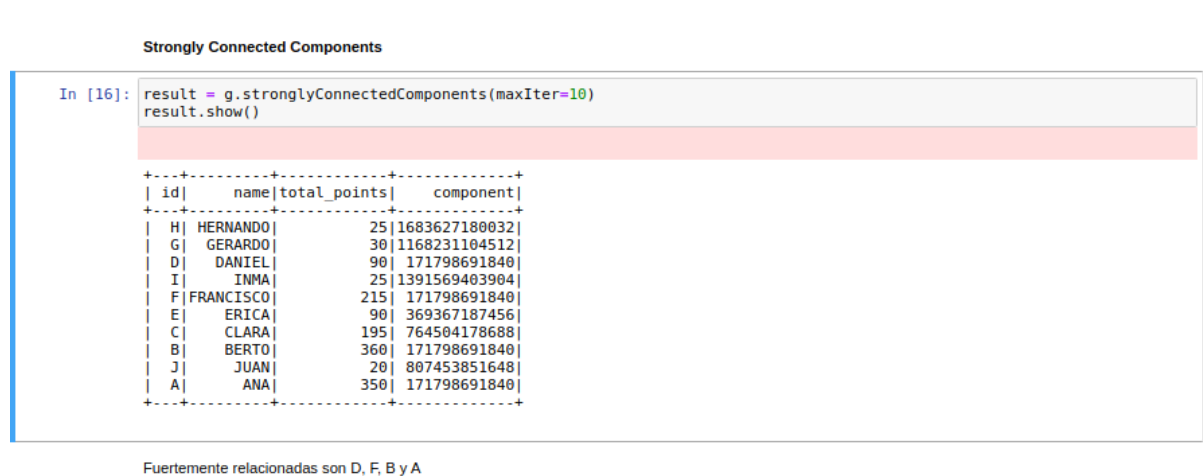


Figura 2.16 Mostrar Componentes fuertemente conectados

Relacionados D, F B, A

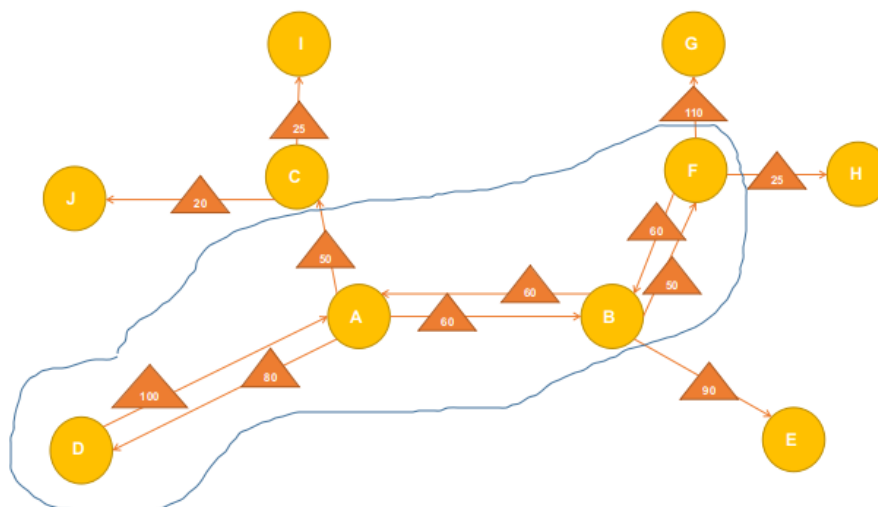


Figura 2.17 Mostrar Componentes fuertemente conectados

Triangle count

Cuenta el número de triángulos para cada nodo en el gráfico y calcula el coeficiente de agrupamiento promedio para la red de nodos resultante. Un triángulo se define como tres nodos que están conectados por tres bordes (a-b, b-c, c-a).

```
results = g.triangleCount()
result.show()
```

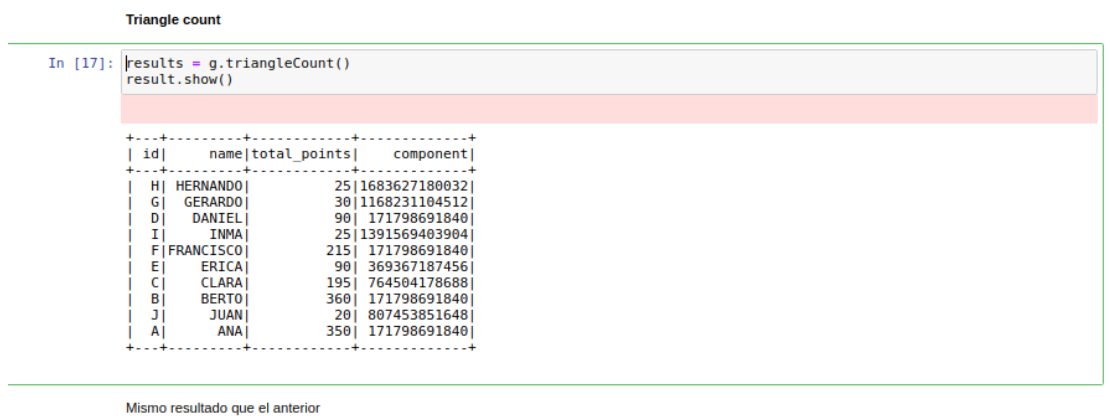


Figura 2.18 Mostrar Triangle Count

En nuestro caso no existe relación.

Shortest paths

Calcula la ruta más corta (ponderada) entre un par de nodos.

```
results = g.shortestPaths(landmarks=['B'])
result.show()
```

Shortest paths

```
In [27]: results = g.shortestPaths(landmarks=['B'])
result.show()
```

id	name	total_points	component
H	HERNANDO	25	1683627180032
G	GERARDO	30	1168231104512
D	DANIEL	90	171798691840
I	INMA	25	1391569403904
F	FRANCISCO	215	171798691840
E	ERICA	90	369367187456
C	CLARA	195	764504178688
B	BERTO	360	171798691840
J	JUAN	20	807453851648
A	ANA	350	171798691840

Figura 2.19 Mostrar rutas cortas

No nos muestra las rutas más cortas

Breadth-first search (BFS)

Recorrer el gráfico desde el nodo raíz y explora todos los nodos vecinos. Luego, selecciona el nodo más cercano y explora todos los nodos inexplorados. El algoritmo sigue el mismo proceso para cada uno de los nodos más cercanos hasta que encuentra el objetivo.

```
paths = g.bfs("name = 'BERTO'", "total_points < 250")
paths.show()
```

Breadth-first search (BFS)

```
In [25]: paths = g.bfs("name = 'BERTO'", "total_points < 250")
paths.show()
```

from	e0	to
{B, BERTO, 360}	{B, F, 50}	{F, FRANCISCO, 215}
{B, BERTO, 360}	{B, E, 90}	{E, ERICA, 90}

Figura 2.20 Mostrar BFS

Subgraphs

Selecciona los nodos que cumplen una serie de normas.

```
g2 = g.filterEdges("relationship = 50").filterVertices("total_points > 30").dropIsolatedVertices()
g2.vertices.show()

g2.vertices.show()
```

Subgraphs

```
In [29]: g2 = g.filterEdges("relationship = 50").filterVertices("total_points > 30").dropIsolatedVertices()
g2.vertices.show()
```

id	name	total_points
F	FRANCISCO	215
B	BERTO	360
C	CLARA	195
A	ANA	350

Figura 2.21 Mostrar Subgráfico

```
g2.edges.show()
```

```
In [30]: g2.edges.show()
```

src	dst	relationship
B	F	50
A	C	50

Figura 2.22 Mostrar Subgráfico

Todas aquellas que tengan una relación de 50 y superior a 30 se total_points

Motif finding

Se conoce como coincidencia de patrones de gráficos. La coincidencia de patrones encuentra algún patrón dentro del gráfico. El patrón es una expresión que se usa para definir algunos vértices conectados.

```
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()
```

Motif finding

```
In [31]: motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()
```

a	e	b	e2
{A, ANA, 350}	{A, D, 80}	{D, DANIEL, 90}	{D, A, 100}
{F, FRANCISCO, 215}	{F, B, 60}	{B, BERTO, 360}	{B, F, 50}
{B, BERTO, 360}	{B, F, 50}	{F, FRANCISCO, 215}	{F, B, 60}
{D, DANIEL, 90}	{D, A, 100}	{A, ANA, 350}	{A, D, 80}
{A, ANA, 350}	{A, B, 60}	{B, BERTO, 360}	{B, A, 60}
{B, BERTO, 360}	{B, A, 60}	{A, ANA, 350}	{A, B, 60}

Figura 2.23 Mostrar Subgráfico

3. DIFERENCIA ENTRE HADOOP Y SPARK

Hadoop y **Spark** son frameworks de código abierto y creados por Apache Software Foundation.

Hadoop es un framework basado en Java que sigue dos sencillos conceptos: el almacenamiento de datos en **Hadoop Distributed File System (HDFS)** y su procesamiento a través de **MapReduce**, también llamado modelo de programación para el procesamiento distribuido de datos.

El **Sistema de Archivos Distribuidos Hadoop (HDFS)** es un sistema diseñado para ejecutarse en hardware común y corriente, económico o conocido como commodity hardware, tiene muchas similitudes con los sistemas de archivos distribuidos existentes, sin embargo, a diferencia de otros, HDFS es altamente tolerante a fallos y está diseñado para ser desplegado en hardware de bajo costo. HDFS proporciona un alto rendimiento de acceso a los datos de la aplicación y es adecuado para aplicaciones que tienen grandes conjuntos de datos.

HDFS tiene una arquitectura maestro/esclavo. Un clúster HDFS consta de un único NameNode, un servidor maestro que gestiona el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los clientes. Además, hay un número de DataNodes, por lo general uno por nodo en el clúster, que gestionan el almacenamiento adjunto a los nodos en los que se ejecutan. HDFS expone un espacio de nombres de sistema de archivos y permite almacenar datos de usuario en archivos.

Internamente, un archivo se divide en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes. NameNode ejecuta las operaciones del espacio de nombres del sistema de archivos como abrir, cerrar y cambiar el nombre de archivos y directorios.

También determina la asignación de bloques a DataNodes. Los DataNodes son responsables de servir las peticiones de lectura y escritura de los clientes del sistema de archivos. Los DataNodes también realizan la creación, eliminación y replicación de bloques a partir de la instrucción del NameNode.

Un trabajo **MapReduce** normalmente divide el conjunto de datos de entrada en bloques independientes que son procesados por las tareas de mapeo de una manera completamente paralela. El framework ordena las salidas de los mapas, que luego se introducen en las tareas de reducción. Normalmente tanto la entrada como la salida del trabajo se almacenan en un sistema de archivos. El framework se encarga de programar tareas, supervisarlas y volver a ejecutar las tareas fallidas.

Principalmente compara y analiza Hadoop y Spark desde cuatro aspectos:

1. **Propósito:** Hadoop es una infraestructura de datos distribuida, que distribuye grandes conjuntos de datos a múltiples nodos en un clúster compuesto por varias computadoras para su **almacenamiento**. Spark es una herramienta especialmente utilizada para procesar macrodatos en almacenamiento distribuido. Spark en sí no almacena datos distribuidos, sino en memoria.

Spark utiliza el cluster computing para su potencia de cálculo (analítica) y su almacenamiento. Esto significa que puede utilizar los recursos de muchos nodos (ordenadores) unidos entre sí para sus análisis. Es una solución escalable que significa que si se necesita más potencia de cálculo, sólo tiene que introducir más nodos en el sistema. Con el almacenamiento distribuido, los enormes conjuntos de datos recogidos para el análisis de grandes volúmenes de datos pueden ser almacenados en múltiples discos duros individuales más pequeños.

Esto acelera las operaciones de lectura y/o escritura, debido al «head», que lee la información de los discos con menos distancia física para desplazarse sobre la superficie del disco. Al igual que con la potencia de procesamiento, se puede añadir más capacidad de almacenamiento cuando sea necesario, el hardware básico y comúnmente disponible (para cualquier disco duro de un ordenador estándar) supone menos costes de infraestructuras.

- 2. Implementación de los dos:** El diseño central del marco de Hadoop es: HDFS y MapReduce. HDFS proporciona almacenamiento para grandes cantidades de datos y MapReduce proporciona cálculos para grandes cantidades de datos. Spark no proporciona un sistema de administración de archivos, pero no solo depende de Hadoop, sino que también puede elegir otras plataformas de sistemas de datos basados en la nube, pero la opción general predeterminada para Spark es Hadoop.

Además Spark está diseñado desde cero para ser fácil de instalar y utilizar –para personas que tiene un mínimo de experiencia en informática-. Ambos están dirigidos a determinados sectores, o con configuración personalizada para proyectos con clientes individuales, así como servicios de consultoría asociados para su creación y funcionamiento.

A diferencia de Hadoop, Spark no viene con su propio sistema de archivos, en lugar de eso, se puede integrar con muchos sistemas de archivos incluyendo de Hadoop HDFS, MongoDB y el sistema S3 de Amazon.

- 3. Velocidad de procesamiento de datos:** Spark tiene las ventajas de Hadoop y MapReduce que son más adecuadas para la minería de datos y el aprendizaje automático que requieren iteración; pero a diferencia de MapReduce, los resultados de salida intermedios del trabajo se pueden almacenar en la memoria. Spark funciona mejor en determinadas cargas de trabajo, ya que proporciona consultas interactivas con conjuntos de datos distribuidos en memoria y también puede optimizar las cargas de trabajo iterativas.

Spark está diseñado para trabajar “In-memory”. Esto significa que transfiere los datos desde los discos duros a memoria principal – hasta 100 veces más rápido en algunas operaciones-.

- 4. Recuperación de la seguridad de los datos:** los datos procesados por Hadoop cada vez que se escriben en el disco, por lo que es intrínsecamente flexible para tratar los errores del sistema; los objetos de datos de chispa almacenados en grupos de datos se denominan elasticidad. En la recopilación de datos distribuidos, estos objetos de datos pueden ser colocado en la memoria o el disco, por lo que Spark también puede completar la recuperación segura de datos.

Comparativa Hadoop y MongoDB:

MongoDB como almacén de datos operativos en tiempo real y Hadoop para el procesamiento y análisis de datos. Algunos diferencias son:

| **Agregación de lotes:** cuando se requiere una agregación de datos compleja MongoDB se queda corto con su funcionalidad de agregación, que no es suficiente para llevar a cabo el análisis de datos. En cambio Hadoop proporciona un potente marco de trabajo que resuelve la situación gracias a su alcance. Para llevar a cabo esta asociación, es necesario extraer los datos de MongoDB (u otras fuentes de datos, si se quiere desarrollar una solución multi-datasource) para procesarlos dentro de Hadoop a través de MapReduce. El resultado puede enviarse de nuevo a MongoDB, asegurando su disponibilidad para posteriores consultas y análisis.

| **Data Warehouse:** en producción, los datos procedentes de una aplicación pueden vivir en múltiples almacenes de datos. Para reducir la complejidad en estos escenarios, Hadoop puede ser utilizado como un almacén de datos y actuar como un depósito centralizado para los datos de las diversas fuentes. En esta situación, podrían llevarse a cabo trabajos MapReduce periódicos para la carga de datos de MongoDB en Hadoop. Una vez que los datos de MongoDB, así como los de otras fuentes, están disponible desde dentro de Hadoop, los analistas de datos tienen la opción de utilizar MapReduce o Pig para lanzar consultas a las bases de datos más grandes que incorporan datos de MongoDB.

| **Procesos ETL:** si bien MongoDB puede ser el almacén de datos operativos para una aplicación, puede suceder que tenga que coexistir con otros almacenes de. En este escenario, es útil alcanzar la capacidad de mover datos de un almacén de datos a otro, ya sea desde la propia aplicación a otra base de datos o viceversa. La complejidad de un proceso ETL excede la de la simple copia o transferencia de datos, por lo que se puede utilizar Hadoop como un mecanismo complejo ETL para migrar los datos en diversas formas a través de uno o más trabajos MapReduce para extraer, transformar y cargar datos en destino. Este enfoque se puede utilizar para mover los datos desde o hacia MongoDB, dependiendo del resultado deseado.

4. PUNTOS CLAVES



No te olvides...

- | Spark MLib sirve para realizar aprendizaje automático.
- | Spark GraphX sirve para **el procesamiento general de gráficos, basándose en la teoría de grafos.**
- | La principal ventaja que presenta Spark frente a Hadoop es que usa la memoria para el procesamiento lo que permite una mayor rapidez de los procesamientos.

